

블록체인 기반 부산 지역 기부 플랫폼



이아영

김동찬

정진규

지도교수 권동현

목 차

1. 서론.....	1
1.1. 연구 배경.....	1
1.2. 기존 문제점.....	1
1.3. 연구 목표.....	1
2. 연구 배경.....	2
2.1. Front-end.....	2
2.1.1. Vue.....	2
2.2. Back-end.....	3
2.2.1. Node.js.....	3
2.2.2. Express.....	3
2.2.3. MongoDB.....	3
2.3. blockchain.....	3
2.3.1. ethereum & truffle.....	3
2.3.2. web3.js.....	3
2.3.3. Metamask & infura.....	4
2.4. 외부 API.....	4
2.4.1. Lightsail.....	4
2.4.2. Pinata ipfs.....	4
2.4.3. alchemy.....	4
3. 연구 내용.....	4
3.1. Front-end.....	4

3.1.1. Vue.js	4
3.2. Back-end.....	10
3.2.1. Node.js.....	10
3.2.2. MongoDB	24
3.3. Blockchain.....	25
3.3.1. Ethereum & Truffle.....	26
3.3.2. Web3.js	29
4. 결론 및 향후 연구 방향	33
5. 개발 일정 및 역할 분담	33

1. 서론

1.1. 연구 배경

기부는 사회적으로 매우 중요한 역할을 한다. 그러나 국내에서는 기부문화가 미처 활성화되어 있지 않아 기부율이 감소하고 있다. 통계청에서 2019년도에 실시한 조사에 따르면 기부 경험이 없는 사람은 74.4%로, 기부 경험이 있는 사람(25.6%)의 약 3배였다.

이러한 문제를 해결하고자 지역 주민들의 유대감을 이용하여 좀 더 기부율을 높이자는 바, 부산 지역 내에서 기부를 할 수 있는 플랫폼을 만들어 블록체인 기술과 스마트 컨트랙트를 활용하여 기부의 투명성과 신뢰성을 보장하고자 한다. 이를 위해, 이더리움 Dapp 플랫폼을 개발하고, 기부 증서를 NFT를 활용한 토큰으로 발행하며, 기부 횟수에 따라 플랫폼에서 만든 NFT를 지급하여 성취감을 충족시키는 등 기부 문화를 활성화하고자 한다.

1.2. 기존 문제점

기존에도 블록체인 기반의 사이트가 이미 존재하였으나 해당 플랫폼에서는 nft 관련 기능이 없었다. 따라서 본 과제에서는 사용자들의 성취감을 만족시켜 나아가 기부를 촉진시키기 위해 nft 발행 기능을 추가하기로 하였다.

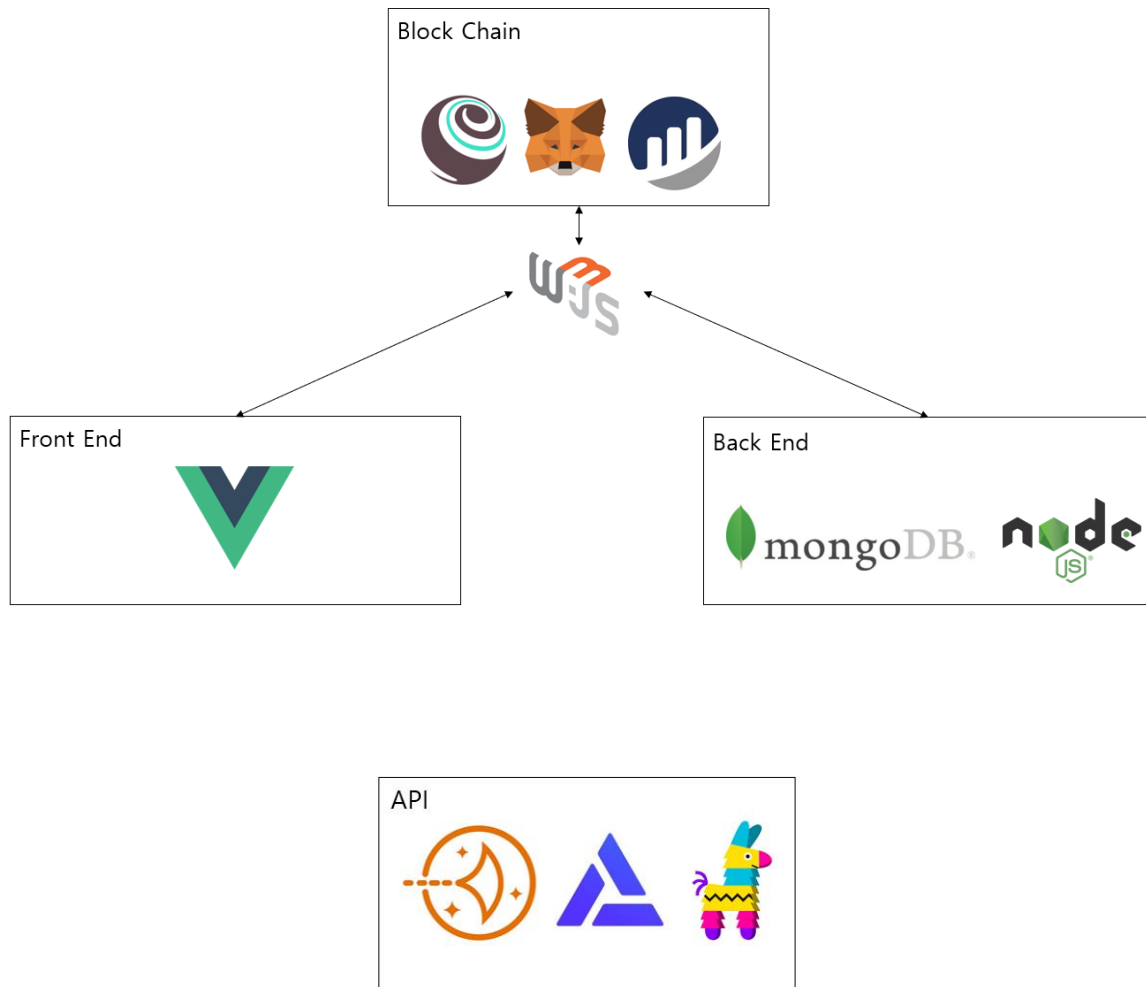
1.3. 연구 목표

본 과제에서는 부산 지역 내 기부를 할 수 있는 이더리움 Dapp 플랫폼을 개발함으로써 블록체인 기술 및 스마트 컨트랙트를 활용하여 기부의 투명성과 신뢰성을 보장하고자 한다.

기부 증서를 NFT를 활용한 토큰으로 발행하여 기부자의 기부 내역을 증명하고, 기부 횟수에 따라 플랫폼에서 만든 NFT를 지급하여 활동적인 기부 문화를 조성하도록 한다. 또한 기부 프로필 생성 및 공유가 가능한 시스템을 구현하여 다른 사용자와 소통할 수 있는 기능을 제공한다.

2. 연구 배경

전체적인 구성도는 아래와 같다.



2.1. Front-end

2.1.1. Vue

Vue.js는 사용자 인터페이스를 만들기 위한 동적 자바스크립트 프레임워크로, 다른 프레임워크들의 사용 언어를 고려하여 선택하였다.

2.2. Back-end

2.2.1. Node.js

확장성 있는 네트워크 애플리케이션을 만들 수 있도록 설계된 비동기 이벤트 주도 JavaScript 런타임이다.

2.2.2. Express

웹 및 모바일 애플리케이션을 위한 일련의 강력한 기능을 제공하는 간결하고 유연한 Node.js 웹 애플리케이션 프레임워크이다. Front-end와 Back-end 모두 Javascript를 사용함으로써 일관되게 개발이 가능하고, 패키지 관리 도구인 npm을 사용함으로써 다양한 라이브러리를 손쉽게 이용할 수 있다.

2.2.3. MongoDB

오픈소스 비관계형 데이터베이스 관리 시스템(DBMS)으로, 테이블과 행 대신 유연한 문서를 활용해 다양한 데이터 형식을 처리하고 저장한다.

mysql과 비교했을때 높은 확장성, 데이터를 문서 형태(스키마 형태 X)로 저장해서 비정형 반정형 정형 데이터 모두 저장할 수 있다. 데이터를 여러 부분집합으로 분할하여 확장할 수 있다.

2.3. blockchain

2.3.1. ethereum & truffle

ethereum이란 블록체인 기반의 오픈소스 플랫폼으로, 스마트 컨트랙트와 탈중앙화된 애플리케이션 구축이 가능하다. 이더리움에서는 클라이언트가 JSON RPC의 형식에 맞춰서 이더리움 노드에 데이터를 요청한다.

Truffle은 이러한 이더리움 기반 DApp 개발에 유틸리티 기능을 제공받을 수 있는 블록체인 프레임워크이다. 스마트 컨트랙트 코드에 대한 컴파일 및 배포 등의 기능을 지원한다.

2.3.2. web3.js

이더리움 네트워크와 상호작용할 수 있는 다양한 메서드를 제공하는 자바스크립트 라이브러리이며, 이더리움 블록체인과 JSON RPC(Remote Procedure Call)를 이용해 상호작용한다.

앞서 서술한 Vue.js와 Node.js에 라이브러리로 들어가 있어 손쉽게 사용이 가능하며, 원

활한 블록체인 상호작용을 할 수 있다.

2.3.3. Metamask & infura

Metamask는 브라우저 확장 프로그램으로 설치할 수 있는 이더리움 블록체인 지갑이다. 메타마스크 사용자는 손쉽게 이더리움 주소로 토큰을 거래할 수 있다.

Infura는 블록체인 테스트넷 서버로, 소스 및 동작의 보안성, 신뢰성, 확장성을 제공하며 이더리움과 IPFS에 접근할 수 있도록 한다. 개발자에게 편의성을 주는 서비스를 제공하여, 본 과제에서는 sepolia 테스트넷을 이용하여 블록체인 서버를 구축하였다.

2.4. 외부 API

2.4.1. Lightsail

빠르게 이미지 업로드를 할 수 있는 외부 이미지 저장소이다.

2.4.2. Pinata ipfs

NFT token URL 설정이 가능하고, 응답속도가 빠른 클라우드 저장소이다.

2.4.3. alchemy

NFT 목록을 불러와 JSON 형태로 반환해주는 API를 지원한다.

3. 연구 내용

3.1. Front-end

3.1.1. Vue.js

```
methods : {  
  getBoardRead() {  
    axios.defaults.headers.common['Access-Token'] = this.$store.state.loginStore.accessToken;  
    axios.get("http://localhost:9000/blockchain/nfts", {  
      headers: {  
        'Authorization' : `Bearer ${this.$store.state.accessToken}`  
      }  
    }).then((res) => {  
      console.log(res);  
      this.boardItem = res.data.data;  
      this.nftItem = this.boardItem.ownedNfts;  
    })  
  }  
}
```

필요한 데이터를 불러오는 부분은 axios를 이용하여 back-end에 요청을 하는 방식으로

구현하였다. 이 때 자신이 회원임을 인증하기 위하여 accesstoken 또한 설정한다.



홈페이지에 처음 들어가면 나오는 화면으로, 우측 상단에 메뉴 및 회원정보 등을 확인할 수 있도록 UI를 구성하였다.

회원가입

성함

성함을 입력하세요.

아이디

아이디를 입력하세요.

패스워드

패스워드를 입력하세요.

계좌

계좌를 입력하세요.

개인 키

키를 입력하세요.

부산 지역 내에 위치하고 있습니다.

회원가입

취소


회원가입의 경우 현재 사용자의 위치를 받아와 부산 지역에 해당되는 위치면 가입이 가능하고, 벗어나는 위치면 가입이 불가하도록 구현하였다.

기부 게시판

쓰기

No	제목	작성자	등록일시
1	모노오르	testid2	2023-09-07 14:48:16
2	기부 캠페인 제목	testid1	2023-09-20 19:00:38
3	testRegister	testRegister1	2023-09-07 18:59:33

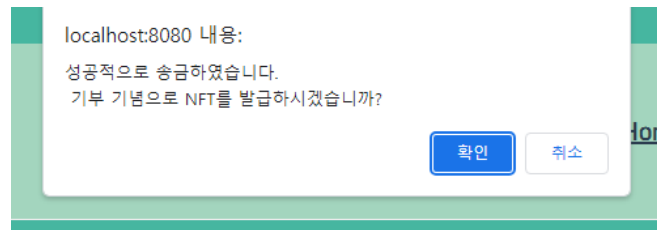
게시글

제목	기부 캠페인 제목
작성자	testid1
작성일자	2023-09-20 19:00:38
최소금액	0
사진	
내용	블록체인 기술과 함께하는 기부문화. 새로운 기술과 함께 행복한 시간을 만끽해보세요.

기부하기

목록

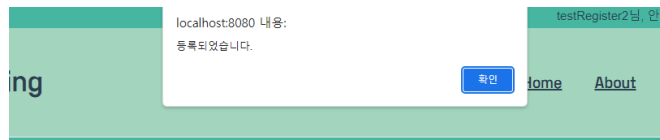
게시글은 작성자가 수정 및 삭제할 수 있다.



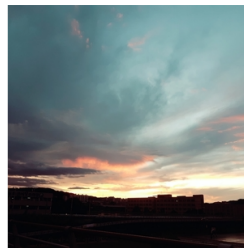
원하시는 기부 금액을 입력해주세요

최소 금액 0

금액



발행자	testRegister2
NFT 이름	<input type="text" value="230929_nft"/>
설명	<div>테스트 nft입니다.</div>
파일 선택	<input type="button" value="파일 선택"/> <div>TY7IzXTOLN...ofileImage.jpg</div>



기부하기 버튼을 누르면 기부 금액을 입력한 후 기부를 할 수 있다. 송금이 성공한다면 일정 횟수에 도달하였을 경우 자동으로 NFT를 발행해주고, 그 후에도 사용자가 원하는 NFT를 발행할 수 있도록 팝업창을 띄운다.

사용자가 NFT 발행을 선택할 경우 페이지를 이동하여 이름과 설명, 이미지를 넣을 수 있도록 구현했다. 확인을 누르면 해당 정보값을 Back-end로 전달하여 nft를 발행할 수 있도록 한다.

testRegister2님, 안녕하세요. 로그아웃 회원수정	
CSE_Chaining	
Home About 기부 게시판	
기부 내역 조회	
기부 날짜	이더스캔 링크
2023-09-08T01:53:44.536Z	https://sepolia.etherscan.io/tx/0xba6b3d9cf9cd3712220bb4537f2e285c9196eacc2ee540baef26ada11c1549f
2023-09-08T01:54:45.096Z	https://sepolia.etherscan.io/tx/0xa52cb04dff515e5e740a3e8fc0478f0a171a6b94e3ecb0c6e38d2f81da568bd7
2023-09-08T02:04:56.596Z	https://sepolia.etherscan.io/tx/0xfa1dc53c9d6589aae9d267a66715b75e4a181c6bea46d124e4102f3c4d5e75e9

마이페이지에서 자신의 기부내역을 확인할 수 있다. 이더스캔 링크를 클릭하여 자신의 기부가 투명하게 이루어졌음을 확인할 수 있다.

testRegister2님, 안녕하세요. [로그아웃](#) [회원수정](#)

CSE_Chaining

[Home](#) [About](#) [기부 게시판](#)

기부 순위

순위	이름	기부 횟수
1	철수	2
2	홍길동	1

기부 횟수에 따른 사용자들의 순위를 보여줌으로써 사용자들의 경쟁심리를 유도한다.

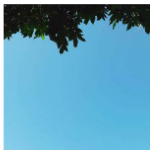
→ ↻
localhost:8080/showNFT

testRegister2님, 안녕하세요. [로그아웃](#)

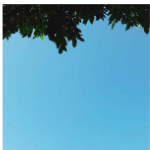
CSE_Chaining
[Home](#)
[About](#)
[기부 게시판](#)

CHAINING PROFILE


철수 / 1번 기부



09180650
1234



1234
1234



ChainingDonate1
1번 기부 토큰

QR생성
이미지로 저장

나만의 기부 프로필을 통해 자신의 닉네임, 기부 횟수, nft 발행 목록을 확인할 수 있다. nft목록을 갤러리처럼 확인할 수 있도록 ui를 구성하였다. 해당 화면에서 사용자는 스크린샷을 통해 프로필을 저장하거나, qr코드 생성을 통해 url을 공유할 수 있다.

3.2. Back-end

3.2.1. Node.js

```
router.get('/', async (req, res) => {  
  try {  
    const filteredBoardList = await Board.find();  
  }  
}  
  
router.get('/:no', async (req, res) => {  
  try {  
    const post = await Board.findOne({ no: req.params.no });  
    const board = new Board();  
    const savedBoard = await board.save();
```

전체 게시글 조회를 위해 Board.find()를 사용한다. 이 함수는 MongoDB에서 Board에 해당하는 모든 문서를 찾아 배열로 반환한다. 또한 await 키워드를 사용하여 비동기 작업을 기다린다.

특정 게시글 조회를 위해서는 Board.findOne() 메소드를 사용하여 특정 조건(no가 req.params.no와 일치하는 경우)에 맞는 문서를 찾는다.

```
const board = new Board();  
const savedBoard = await board.save();
```

새로운 Board 인스턴스를 생성하고, .save() 메소드를 호출하여 MongoDB에 저장함으로써 게시글을 생성할 수 있다.

```
router.put('/:no', async (req, res) => {  
  const { subject, content, minPrice, imageLink } = req.body;  
  const board = await Board.findOne({ no: req.params.no });
```

게시글 수정을 위해서는 먼저 .findOne()으로 수정할 문서를 찾고, 필드를 수정한 후 .save()로 변경사항을 저장한다.

```
router.delete('/:no', async (req, res) => {
  const result = await Board.findOneAndDelete({ no: req.params.no });
```

게시글 삭제를 위해서는 .findOneAndDelete() 메소드를 사용하여 특정 조건에 맞는 문서를 찾아서 삭제한다.

```
{
  _id: ObjectId('64f989f22f9d4bb1a1f8c1e6'),
  no: 2,
  subject: "기부 캠페인 제목",
  content: "블록체인 기술과 함께하는 기부문화.  
새로운 기술과 함께 행복한 시간을 만끽해보세요.",
  writer: "testid1",
  writedate: "2023-09-20 19:00:38",
  minPrice: 0,
  imageLink: "https://chaining.s3.ap-northeast-2.amazonaws.com/1695204010889-32595.j...",
  __v: 0
}
```

위와 같이 mongoDB에 게시글이 저장되는 걸 확인할 수 있다.

```
const express = require('express')
const cors = require('cors');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
```

App.js의 기본 구성으로 Express, cors(cross-origin resource sharing), mongoose(mongodb와 연결을 위한 라이브러리), dotenv(env에서 환경변수 로드하기 위한 라이브러리)와 같은 모듈을 import하였다.

```
const app = express()
const port = process.env.PORT;
app.use(express.json());
app.use(cors());
app.use(express.static('uploads'));
```

app 객체를 생성하고 포트 정보를 가져오며, express.json()을 통해 JSON 형태의 요청 본문을 파싱한다. 또한 CORS 미들웨어를 사용하여 다른 도메인에서의 요청을 허용하도록 한다. 마지막으로 uploads 디렉토리를 정적 리소스로 사용하도록 설정한다.

회원가입 API [🔗](#)

Request [🔗](#)

POST /members/sign-up

[Body]

- id : 계정 (string)
- name : 이름 (string)
- password : 비밀번호 (string)
- account : 계좌번호 (string)
- privateKey : 개인키 (string)

Response [🔗](#)

```
{
  "success": true,
  "data": {
    "transactionCount": 0,
    "_id": "64f954a2908c93344b007603",
    "name": "이름",
    "id": "아이디",
    "password": "비밀번호",
    "myAccount": "계좌",
    "privateKey": "개인키",
    "__v": 0
  }
}
```

```
router.post('/sign-up', async (req, res) => {
  const { name, id, password, myAccount, privateKey } = req.body;
  const member = new Member();
  member.name = name;
  member.id = id;
  member.password = password;
  member.myAccount = myAccount;
  member.privateKey = privateKey;
  const savedMember = await member.save();
  res.json({ success:true, data: savedMember });
})
```

회원가입의 경우 POST /sign-up 경로로 요청이 오면 처리한다. 클라이언트로부터 받은 데이터를 Member 모델을 사용하여 MongoDB에 저장하도록 구현하였다.

Refresh Token API [↗](#)

Request [↗](#)

POST /members/refresh [↗](#)

[Body]

- id : 계정 (string)
- accessToken : Access Token (string)
- refreshToken : Refresh Token (string)

Response [↗](#)

```
{
  "success": true,
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZW11ZXJJZCI6InRlc3RpZDEiLCJtZW11ZXJ0YW11Ijo7ZmN6r1464-ZI."
}
```

```
router.post('/refresh', async (req, res) => {
  const memberId = req.body.id;
  const accessToken = req.body.accessToken;
  const refreshToken = req.body.refreshToken;

  try {
    var memberItem = await Member.findOne({ id: memberId });

    if (memberItem != null) {
      let refreshPayload = "";
      let errorMessageRT = "";

      // Refresh-Token Verify
      try {
        refreshPayload = await new Promise((resolve, reject) => {
          jwt.verify(refreshToken, process.env.JWT_SECRET,
            (err, decoded) => {
              if (err) {
                reject(err);
              } else {
                resolve(decoded);
              }
            }
          );
        });
      } catch(err) {
        errorMessageRT = err;
      }

      console.log("Refresh-Token Payload : ");
      console.log(refreshPayload);
      console.log("Refresh-Token Verify : " + errorMessageRT);

      let accessPayload = "";
      let errorMessageAT = "";
    }
  }
});
```

POST /refresh 경로로 요청이 오면 토큰을 갱신한다.

[Header]

- Access-Token : JWT Access Token (String)


```
router.get('/me', authMiddleware, async (req, res) => {
  try {
    const member = await Member.findOne({ id: req.tokenInfo.memberId });

    if (!member) {
      res.status(404).json({ success: false, errormessage: 'not found' });
      return;
    }

    res.json({ success: true, data: member });
  } catch (error) {
    res.status(500).json({ success: false, errormessage: 'internal server error' });
  }
});
```

GET /me 경로로 요청이 오면 현재 인증된 사용자의 정보를 반환한다.

사용자 정보 수정 [↗](#)

Request [↗](#)

PUT /members [↗](#)

[Header]

- Access-Token : JWT Access Token (String)

[Body]

- id: 계정 (string, Optional)
- password: 비밀번호 (string, Optional)
- name: 이름 (string, Optional)

Response [↗](#)

```
{  
  "success": true  
}
```

[↗](#)

게시글 전체 조회 API [↗](#)

Request [↗](#)

GET /boards [↗](#)

Response [↗](#)

```
{  
}
```

[↗](#)

```
router.put('/', authMiddleware, async (req, res) => {  
  const { id, password, name } = req.body;  
  const memberId = req.tokenInfo.memberId;  
  const member = await Member.findOne({ id: memberId });  
  if (id) {  
    member.id = id;  
  }  
  if (password) {  
    member.password = password;  
  }  
  if (name) {  
    member.name = name;  
  }  
  await member.save();  
  res.json({ success:true });  
})
```

PUT / 경로로 요청이 오면 인증을 거친 후 회원 정보를 수정한다.

게시글 전체 조회 API

Request

GET /boards

Response

```
{  
}
```

특정 게시글 조회 API

Request

GET /boards/:no

[path params]

- no : 게시글 번호 (number)

[body]

- subject : 제목 (string)
- content : 내용 (string)

Response

```
{  
}
```

```
router.get('/', async (req, res) => {  
  try {  
    const filteredBoardList = await Board.find();  
    const sortedBoardList = await sorting(filteredBoardList, req.query.sortby);  
  
    const paginationInfo = await pagination(sortedBoardList, req.query);  
  
    const startIndex = (paginationInfo.pageNo - 1) * paginationInfo.countPerPage;  
    const endIndex = startIndex + paginationInfo.countPerPage;  
  
    const paginatedBoardList = sortedBoardList.slice(startIndex, endIndex);  
  
    res.json({ success: true, data: paginatedBoardList, paginationInfo });  
  } catch (error) {  
    console.log(error);  
    res.status(500).json({ success: false, errorMessage: 'internal server error' });  
  }  
});  
  
router.get('/:no', async (req, res) => {  
  try {  
    const post = await Board.findOne({ no: req.params.no });  
  
    if (!post) {  
      res.status(404).json({ success: false, errorMessage: 'not found' });  
      return;  
    }  
    res.json({ success: true, data: post });  
  } catch (error) {  
    res.status(500).json({ success: false, errorMessage: 'internal server error' });  
  }  
})
```

GET / 경로로 요청이 오면 게시글 목록을 반환한다.

클라이언트로부터 받은 데이터를 Board 모델을 사용하여 MongoDB에서 조회한 후 응답한다.

GET /:no 경로로 요청이 오면 특정 게시글을 반환한다.

클라이언트로부터 받은 게시글 번호를 사용하여 MongoDB에서 조회한 후 응답한다.

게시글 작성 API [🔗](#)

Request [🔗](#)

POST /boards [🔗](#)

[Header]

- Access-Token : JWT Access Token (String)

[body]

- subject : 제목 (string)
- content : 내용 (string)
- imageLink : 이미지 링크 (string)

Response [🔗](#)

```
{
  "success": true
}
```

```
router.post('/', authMiddleware, async (req, res) => {
  const lastBoard = await Board.findOne().sort({ _id: -1 }).exec();
  const board = new Board();
  if (lastBoard) {
    board.no = lastBoard.no + 1;
  } else {
    board.no = 1;
  }
  board.subject = req.body.subject;
  board.content = req.body.content;
  board.writer = req.tokenInfo.memberId;
  const currentDate = dayjs().format('YYYY-MM-DD HH:mm:ss');
  board.writedate = currentDate;
  board.minPrice = req.body?.minPrice | 0;
  board.imageLink = req.body.imageLink;
  const savedBoard = await board.save();
  res.json({ success: true, data: savedBoard });
})
```

POST / 경로로 요청이 오면 새로운 게시글을 등록한다. 인증 미들웨어를 거친 후, 클라이언트로부터 받은 데이터를 Board 모델을 사용하여 MongoDB에 저장한다.

특정 게시물 수정 API

Request

PUT /boards/:no

[Header]

- Access-Token : JWT Access Token (String)

[path params]

- no : 게시물 번호 (number)

[body]

- subject : 제목 (string)
- content : 내용 (string)
- minPrice : 최소 금액 (number)
- imageLink : 이미지 링크 (string)

Response

```
{
  "success": true
}
```

```
router.put('/:no', async (req, res) => {
  const { subject, content, minPrice, imageLink } = req.body;
  const board = await Board.findOne({ no: req.params.no });
  if (!board) {
    res.json({ success: false, errorMessage: '존재하지 않는 게시물입니다.' });
  }
  if (subject) {
    board.subject = subject;
  }
  if (content) {
    board.content = content;
  }
  if (minPrice) {
    board.minPrice = minPrice;
  }
  if (imageLink) {
    board.imageLink = imageLink;
  }
  const currentDate = dayjs().format('YYYY-MM-DD HH:mm:ss');
  board.writedate = currentDate;
  board.save();

  res.json({ success: true, data: board });
})

router.delete('/:no', async (req, res) => {
  const result = await Board.findOneAndDelete({ no: req.params.no });
  if (!result) {
    res.json({ success: false, errorMessage: '존재하지 않는 게시물입니다.' });
    return;
  }
  res.json({ success: true })
})
```

PUT /:no 경로로 요청이 오면 해당 게시글을 수정한다.

클라이언트로부터 받은 데이터를 사용하여 MongoDB에서 해당 게시글을 수정한 후 응답한다.

특정 게시물 삭제 API

Request

DELETE /boards/:no

[Header]

- Access-Token : JWT Access Token (String)

[path params]

- no : 게시물 번호 (number)

Response

```
{
  "success": true
}
```

```
router.delete('/:no', async (req, res) => {
  const result = await Board.findOneAndDelete({ no: req.params.no });
  if (!result) {
    res.json({ success: false, errorMessage: '존재하지 않는 게시물입니다.' });
    return;
  }
  res.json({ success: true })
})
```

DELETE /:no 경로로 요청이 오면 해당 게시글을 삭제한다.

클라이언트로부터 받은 게시물 번호를 사용하여 MongoDB에서 해당 게시글을 삭제한 후 응답한다.

이미지 업로드 API

Request

POST /upload

[주의]

- multipart/form-data로 전송해야 함

[form-data]

- photos : 사진 (File[])

Response

```
{
  "success": boolean,
  "photos": [string]
}
```

```
router.post('/', upload.array("photos", 10), async (req, res) => {  
  console.log(req.files);  
  const photos = req.files.map(file => file.location);  
  res.send({ success: true, photos });  
})
```

POST / 경로로 요청이 오면 파일을 S3에 업로드한다.

upload.array 미들웨어를 사용하여 여러 파일을 한 번에 최대 10개까지 업로드할 수 있다.

기부 순위 조회 API

Request

GET /transactions/rank

[Header]

Response

```
{
  "success": true,
  "data": [
    {
      "member": {
        "_id": "64fa7e5f9a9b789979052898",
        "transactionCount": 22,
        "name": "철수",
        "id": "testRegister2",
        "password": "testRegisterPwd1",
        "myAccount": "0xd3781f83cfD979a8C8001Db8f6FAf684686ee2d",
        "privateKey": "63484d87cbce5803a4fef3b6ac1ae6aaf9c84cef43048ae3dd3fc7bb4af3a3f1",
        "_v": 0,
        "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZW1iZXJJZCI6InRlc3RSZWdpY3RlcjIiLCJpYXQiOiJE20TQ1Mz",
      },
      "rank": 1
    }
  ]
}
```

```
// 기부 순위 조회
router.get('/rank', async (req, res) => {
  try {
    const members = await Member.find().sort({ transactionCount: -1 })

    const results = members.map((member, index) => {
      return {
        member,
        rank: index + 1
      }
    })

    res.json({ success: true, data: results });
  } catch (error) {
    res.status(500).json({ success: false, errorMessage: 'internal server error' });
  }
})
```

GET /rank 요청을 처리하여 모든 사용자의 기부 순위를 반환한다.

Member.find().sort({ transactionCount: -1 })를 통해 transactionCount 필드를 기준으로 내림순 정렬한 회원 목록을 가져온다.

map() 함수를 사용하여 순위와 함께 반환될 데이터를 구성한다.

이미지를 업로드할 때 amazon 버킷을 사용하여 업로드한다.

업로드를 진행한 후 amazon webapp에서 확인해보면 정상적으로 잘 업로드된 것을 확인

할 수 있다.

3.2.2. MongoDB

본 과제에서는 MongoDB ODM(Object Document Mapping)중 가장 유명한 라이브러리인 mongoose를 사용했다.

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
dotenv.config(); // .env 파일에서 환경 변수 로드
```

Mongoose 라이브러리를 불러오고, .env를 통해서 MongoDB URI와 같은 정보를 로딩했다.

```
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => console.log("MongoDB 연결 성공..."))
.catch((err) => console.log(err));
```

Mongoose.connect() 함수를 사용하여 MongoDB에 연결했다.

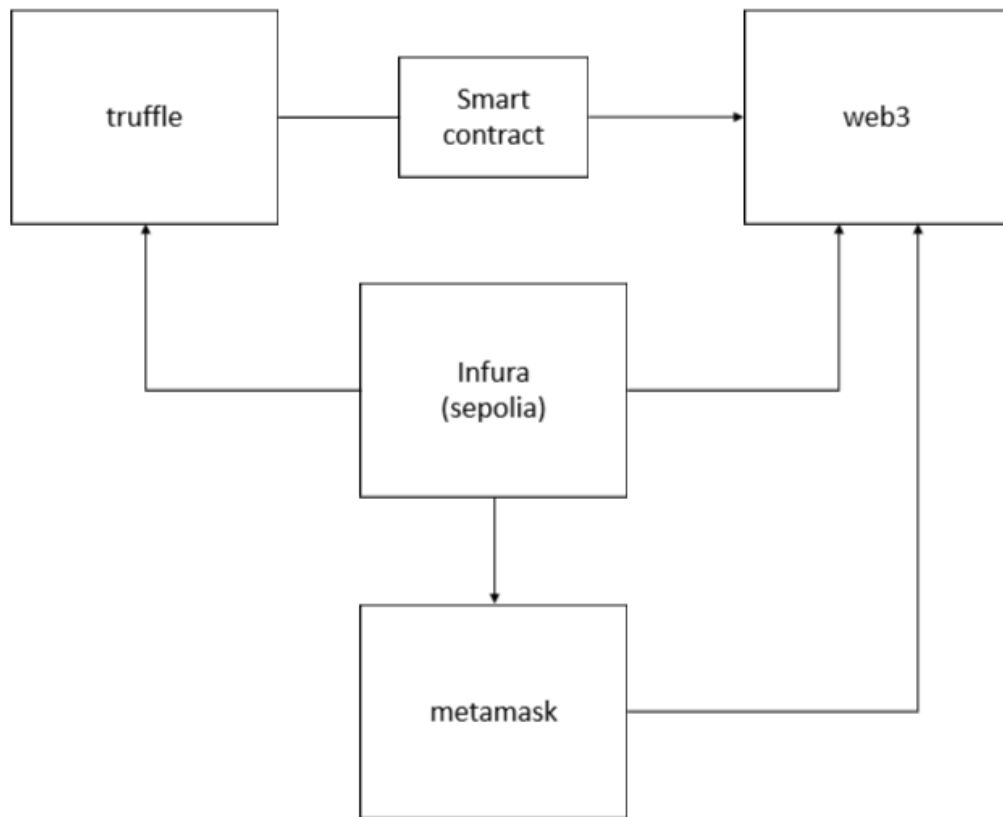
```
const mongoose = require('mongoose');
const boardSchema = new mongoose.Schema({
  no: Number,
  subject: String,
  content: String,
  memberId: String,
  writer: String,
  writedate: String,
  minPrice: Number,
  imageLink: String,
});
module.exports = boardSchema;
```

위와 같은 방식으로 게시글 스키마를 정의했다.

```
const { Board } = require("../models");
```

../models 디렉토리에 정의된 mongoose 모델을 불러와 mongoDB와 상호작용한다.

3.3. Blockchain



전체적인 구조는 위와 같다. infura에서 제공하는 블록체인 sepolia 테스트넷을 중심으로, truffle과 infura를 연결한 후 smart contract를 생성한다. 만들어진 컨트랙트를 web3가 이용한다. 또한 블록체인 지갑인 metamask와 infura를 연결하여 지갑을 생성할 수 있도록 하고, 이 지갑을 web3에서 이용하여 송금 혹은 잔액 확인 등의 기능을 실행할 수 있도록 한다.

3.3.1. Ethereum & Truffle

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "5777"
    },
    sepolia: {
      provider: () => new HDWalletProvider(MNEMONIC, 'https://sepolia.infura.io/v3/'),
      network_id: 11155111,
      gas: 4000000,
      gasPrice: 10000000000,
      networkCheckTimeout: 100000,
    },
  },
  compilers: {
    solc: {
      version: "0.8.7",
      optimizer: {
        enabled: true,
        runs: 200
      }
    }
  }
},
```

기존에는 ganache 테스트넷을 사용하여 개발을 진행하였다. 하지만 이는 로컬 테스트넷이라는 한계가 존재하였기 때문에 Infura의 sepolia 테스트넷으로 변경하여 개발을 진행하였다.

네트워크 이름

sepolia 네트워크

새 RPC URL

https://sepolia.infura.io/v3/

체인 ID ⓘ

11155111

통화 기호

ETH

블록 탐색기 URL (옵션)

https://sepolia.etherscan.io/

 계정 3 \$5,221.56 USD ⋮
0xd37...ee2d 3.122 ETH

 Account 2 \$4,169.32 USD ⋮
0x560...b426 2.492 ETH

가져옴

해당 서버에 맞게 metamask 네트워크도 함께 세팅해주었다.

```

// Method for calling the contract method
if (window.ethereum) {
  this.web3 = new Web3(window.ethereum);
} else if (typeof window.web3 !== 'undefined') {
  this.web3 = new Web3(window.web3.currentProvider);
} else {
  alert('No web3 instance injected, using local web3.')
  //메타마스크를 설치하라는 alert 띄워줄 것
}
if (this.web3) {
  this.account = await this.web3.eth.requestAccounts(); //이렇게 하면 계좌 리스트가 불러와지고
  this.my_account = this.account[0]; //맨 첫번째 요소를 불러와서 계좌에 셋팅
  this.valance = await this.web3.eth.getBalance(this.account[0]); //본인 계좌 잔고 함수
  console.log(this.account);
  console.log(this.valance);

  let result = Number(this.valance);
  result = result/Math.pow(10,18);
  result = result.toFixed(4);

  this.valance = result;

  this.contractResult = '현재 잔고는 ' + result.toString() + 'ETH 입니다.';
}
}

```

Window.ethereum 함수를 사용하여, metamask와 연결할 수 있도록 구현하였다.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.7;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

contract Nft is ERC721, Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() public ERC721("Chaining", "chainingNFT") {}

    function mintNFT(address recipient, string memory tokenURI)
        public returns (uint256)
    {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(recipient, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```

NFT 발행 컨트랙트로, '대체 불가능성'을 지니고 있는 ERC-721을 사용하였다. 필요한 정보값을 받아오면 그 정보값을 토대로 Chaining Collection NFT를 만들어낸다.

3.3.2. Web3.js

1) 송금

```
web3.eth.getTransactionCount(my_a var BigInt: BigIntConstructor
const hex = web3.utils.toHex(tx (value: string | number | bigint | boolean) => bigint
const value = web3.utils.toHex(BigInt(web3.utils.toWei(send_eth, "ether")))

const txObject = {
  'nonce': hex, // Transaction의 발동 횟수
  'from': my_account, // 발신자
  'to': receive_account, // 수신자
  'value': value,
  'gasLimit': web3.utils.toHex(21000), // 가스 한도
  'gasPrice': web3.utils.toHex(BigInt(web3.utils.toWei("0.00000001", "ether"))),
  'chainId': 11155111,
};

const tx = new Tx(txObject);
tx.sign(privateKeyBuffer);

const serializedTx = tx.serialize();

const raw = '0x' + serializedTx.toString('hex');
web3.eth.sendSignedTransaction(raw).then(async (txHash) => {
  console.log("-----")
  const url = 'https://sepolia.etherscan.io/tx/' + txHash.transactionHash
  console.log(txHash);
});
```

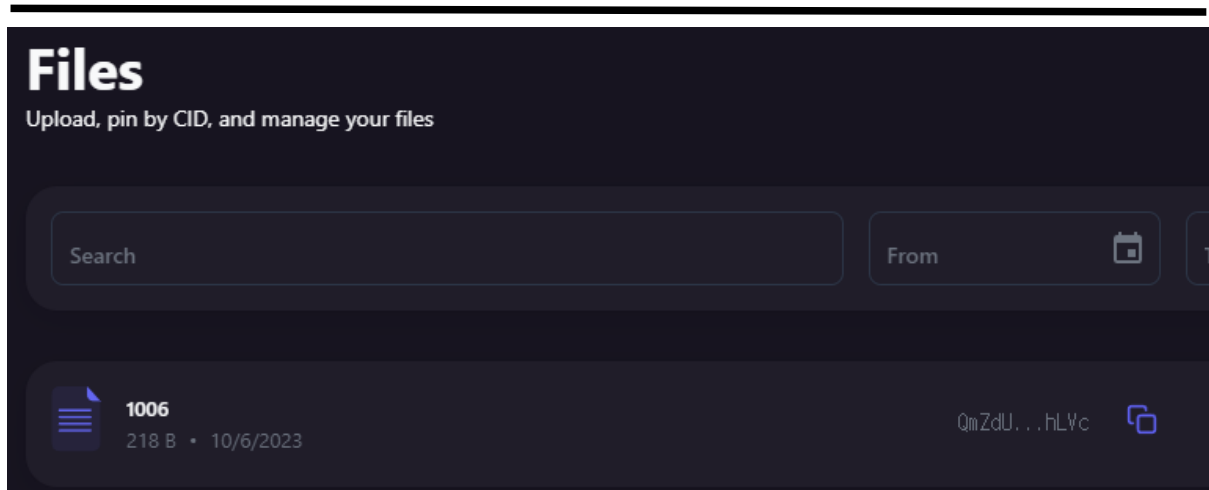
우선 transactionCount 함수를 사용하여 거래 횟수를 측정한 뒤 nonce 값을 받아오고, txObject를 생성하여 거래 사전 정보를 작성한다. 여기서 가스 가격을 작성할 때 int가 표현할 수 없는 값을 감안하여 자바스크립트에서 제공하는 bigint를 사용하여 작성한다. 또한 가스 한도와 가격은 잔고나 속도 등을 고려하여 적절한 값을 할당해주었다.

이후 개인키로 서명을 한 후 거래를 진행한다. 거래가 완료되면 반환된 거래 고유의 해시값을 통해 이더스캔 url을 추출해낸 후 데이터베이스에 저장할 수 있도록 한다.

2) NFT 발행

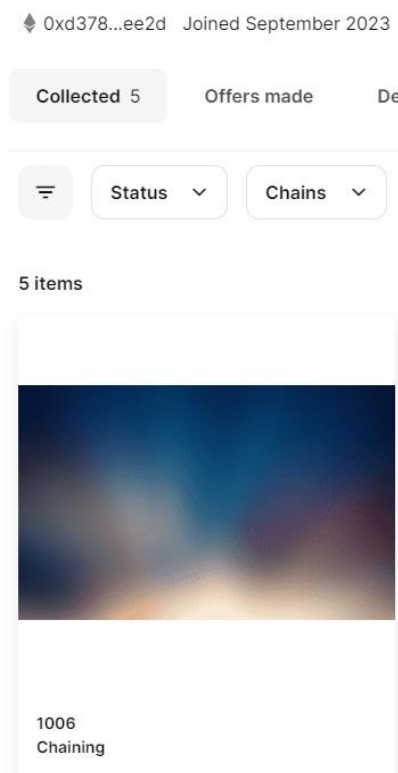
```
const pinata_upload2 = async (NFTname, description, img, my_account) => {  
  var createJson = {  
    name : NFTname,  
    description : description,  
    image : img,  
  }  
  var jsonData = JSON.stringify(createJson);  
  var filePath = `./upload/${NFTname}`;  
  
  fs.writeFileSync(filePath, jsonData, (err) => {  
    console.log(err);  
  })  
  let metaData = await ipfs.pinataUpload(filePath);  
  
  var metaData2 = `https://gateway.pinata.cloud/ipfs/${metaData}`;  
  
  console.log('!!!-----', metaData2);  
  
  web3.eth.getTransactionCount(contractOwner).then((txCount) => {  
    const hex = web3.utils.toHex(txCount);  
  
    const txObject = {  
      'nonce': hex,  
      'to': contractAddress,  
      'from': my_account,  
      'gasLimit': web3.utils.toHex(1000000),  
      'gasPrice': web3.utils.toHex(BigInt(web3.utils.toWei("0.00000001", "ether"))),  
      'chainId': 11155111,  
      'data': MyContract.methods.mintNFT(my_account, metaData).encodeABI(),  
    };  
    const tx = new Tx(txObject);  
    tx.sign(privateKeyBuffer);  
  
    const serializedTx = tx.serialize();  
  
    const raw = '0x' + serializedTx.toString('hex');  
    web3.eth.sendSignedTransaction(raw).then((txHash) => {  
      console.log(txHash);  
    })  
  });  
};
```

사용자의 입력값을 토대로 정보를 구성한다. 미리 lightsail에 업로드해둔 이미지를 metadata를 pinataCloud에 업로드한다. 업로드가 끝나면 해당 metadata를 이용하여 nft 발급을 한다. 여기서 미리 구현해놓았던 스마트 컨트랙트를 실행시켜 발행한다.



```
{ "name": "1006", "description": "1006test", "image": "https://chaining.s3.ap-northeast-2.amazonaws.com/1696570763657-png-transparent-gradient-desktop-color-background-miscellaneous-blue-atmosphere-thumbnail.png" }
```

먼저 pinata cloud 에 위와 같이 metadata 가 저장된다. 해당 metadata 를 이용하여 컨트랙트를 진행한다.



이후 외부 사이트를 이용하여 확인해보면 NFT가 제대로 발급되어있는 것을 확인할 수 있다.

3) NFT 목록 확인

```
const memberId = req.tokenInfo.memberId;
const member = await Member.findOne({ id: memberId });
const private_key = member.privateKey;
const my_account = member.myAccount;
const privateKeyBuffer = Buffer.from(private_key, "hex");

const api_key = [REDACTED]

const settings = {
  apiKey: api_key,
  network: Network.ETH_SEPOLIA,
  withMetadata: 'true'
};

console.log(settings.apiKey);

const alchemy = new Alchemy(settings);

alchemy.nft.getNftsForOwner(my_account).then(result => {
  console.log("GETNFT_RESULT")
  console.log(result)
  console.log(result.ownedNfts[0].contract)
  result.ownedNfts = result.ownedNfts.filter(nft => {
    return nft.contract.name === 'Chaining'
  });
  const convertedResult = {
    ownedNfts: result.ownedNfts.map(nft => {
      return {
        title: nft.title,
        description: nft.description,
        media: nft.media[0]?.gateway
      }
    })
  }
  res.send({ success: true, data: convertedResult });
});
```

Alchemy api를 이용해 nft 발행 목록을 불러온 뒤, 그 중 본 플랫폼에서 발급받은 nft만 골라 확인할 수 있도록 필터링 하여 정리한다.

4. 결론 및 향후 연구 방향

이더리움 기반으로 블록체인 기반의 기부 플랫폼을 구현함으로써 투명성있는 기부 플랫폼을 설계할 수 있었다. 하지만 본 과제의 한계점은 아래와 같다.

- 1) 개인정보를 수집하는 과정에서 보안 처리를 따로 해주지 않아 개인정보 유출 가능성이 존재한다.
- 2) 기부 절차의 투명성은 보장되었으나 정작 기부를 받는 기관에 대한 신뢰성은 보장할 수 없다.

따라서 추후 개인정보 수집에 대한 보안에 대해 좀 더 신경쓰고, 블록체인에 대한 성능을 개선하고자한다. 또한 부산 지역 플랫폼이 주제인 만큼 부산의 이미지를 좀 더 잘 드러낼 수 있도록 연구를 진행할 예정이다.

5. 개발 일정 및 역할 분담

	5				6				7				8				9			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
기획																				
블록체인 관련 스터디																				
웹사이트 UI 설계																				
블록체인 개발																				
DB 구축																				
웹사이트 개발																				
테스트 및 디버깅																				
최종 점검 및 발표 준비																				

이아영	블록체인 알고리즘 개발 NFT 지갑 연동 및 관리 웹사이트 개발
김동찬	데이터베이스 구축 및 관리 서버 구축 및 관리
정진규	웹사이트 개발

2023 년 전기 산학협력 프로젝트 멘토 의견서

1. 지도개요

팀 명	Chaining		
과 제 명	블록체인 기반 부산 지역 기부 플랫폼		
협력기관	스마트엠투엠		
참여학생	이름	전화번호	이메일
	김동찬	010-2842-1490	ehdcks1224@naver.com
	이아영	010-3281-7234	sisami00@pusan.ac.kr
	정진규	010-2237-3969	jjk3969@pusan.ac.kr
참여교수명	권동현		

2. 세부 지도 내용

- 본 과제에서 블록체인이 필요한 이유가 무엇인지?
- ex) 글에서 제시한 블록체인을 통해 투명성과 신뢰성을 보장하기 위함이라고 한다면, 기부 문화의 폐쇄성에 의해 발생하는 여러 문제점들을 배경으로 소개하고 이를 방지할 수 있다는 취지의 스토리가 필요함
- 기존 블록체인 기부 플랫폼이 있는데 이와 비교했을 때의 차별성이 필요해보임
- 현재 과제명은 부산 지역을 대상으로한다는 지역특화에 대한 아이디어나, NFT 기부 증서에 대한 혜택 (기부자들에 대한 투자를 받아서 연계하는 등... BM 측면에서의 새로운 기능 플로우) 등... 기존 서비스와는 다른 장점이나 차별성/혁신성이 보이는 아이디어를 포함하는게 졸업과제의 취지와 맞는 것으로 보임
- 타플랫폼 참조: <https://givecherry.org/> 블록체인 기반 기부 플랫폼 '체리'
- 설계 구조 점검이 필요
- 현재 구현 관점에서 블록체인과 설계는 Front<->Back<->Blockchain으로 되어 있는데, 실질적으로 진행된 내용을 보면 Front<->Blockchain/ Front<->Back으로 보임. 설계 아키텍처 그림 검토가 필요해보임
- Metamask를 활용한다고 하더라도, 현재 제안한 기부 플랫폼과 같이 Service Provider가 존재하는 DApp의 특성상 Backend에서 기부에 대한 관리를 수행하여야함. 현재 보고서 상에서의 구현된 블록체인 스마트 컨트랙트는 단순히 토큰을 송금만하는 형태인데... DApp에 대한 서비스를 Front/Back을 염두하여 개발한다면 Service Provider는 이미 구성된 형태로 해당 Provider에 의한 중앙화된 직접 관리가 필요해보임
- 기능 측면에서 전체 진행도 점검 필요
- 구현을 수행한다면, 기본적으로 초기 시나리오 작성/기술 스택/기능 요구사항 정의/아키텍처 설계 등의 과정으로 진행될텐데 기능에 대한 요구사항들이 현재 제시한 구현 내용과 매핑되지 않음. 플로우 차트 상에 보이는 기능들과는 다르게 중간 보고에 작성된 기능들은 단순히 '송금 스마트 컨트랙트' + '블록체인 연계' + '게시판 글쓰기' 정도만 구현된것으로 보임. 전체 일정상 기한내 개발이 가능할지, 집중이 필요해보임

- 기부 시나리오를 기본으로 한다면, 기부하기를 위해서는 송금 + NFT 발행(or 업데이트) 기능 + 기부 단체 연계, 기부 조회하기의 경우에는 NFT 조회 + 단체 증명서 검증/연계 ... 등등의 기능들을 현재 구현해야함
- 기부에 대한 관리를 수행한다면, 기부에 대한 데이터가 블록체인 상에 있더라도 Query 측면에서 빠른 조회를 위해 StateDB 를 운영하거나, Backend 에 별도로 기록해야하는 등 최소 기능들을 빠르게 구현해야함
- (NFT 의 경우 등급별 구분이 가능해야하기 때문에, 기부 횟수에 따라 추가 발행을 하거나 Dynamic NFT 와 같은 새로운 유형을 도입해야할 것으로 보이는데... 좀더 면밀한 기술 검토와 설계 필요)

전체적으로 블록체인의 필요성/철학에 대한 명확한 스토리텔링을 기반으로 차별성을 강조할 수 있는 아이디어가 포함되고 개발에도 반영되는 것이 이상적인 방향으로 보임. 특히 기부 문화의 흐름을 추적한다고 하더라도, 데이터 검증/신뢰성 관점에서 오라클 문제 등 해결해야할 다양한 키워드 요소들이 있는데 이런 부분들이 현재 프로젝트에 추가된다면 좋을 것으로 보임

다만, 현재 명시한 기능들 대비 구현에 대한 부분이 미비하여 진행상황에 맞추어 기본 기능과 보여줄 수 있는 부분들, 개념적으로 중요한 핵심 기능을 우선 구현하는 등의 개발 전략을 통해 완성도를 효율적으로 높일 필요가 있다고 판단됨

또한, 만약 여유가 된다면 블록체인 플랫폼을 변경해야할 것으로 보임. 이더리움의 경우 트랜잭션의 비용이 비싸기 때문에, 서비스 상용화 측면에서는 고려할 수 없어 구현/설계상으로 100% 이식성/호환성을 제공하는 Solana, Polygon 등 Public 블록체인을 통해 NFT 발행 비용이나 TPS/Latency 까지 염두하여 개발한다면 더욱 양질의 결과물이 나올 것으로 사료됨

위 내용을 부산대학교 정보컴퓨터공학부 2023 학년도 전기 산학협력프로젝트 지도내용으로 제출합니다.

멘토링 일시	2023 년 9 월 13 일	시작시간	19 : 00	종료시간	21 : 00
--------	-----------------	------	---------	------	---------

소속: 스마트엠투엠

직급: 팀장

성명: 김명길

(서명)